The Challenges of Trace-Driven Wi-Fi Emulation

Mohammad Imran Syed, Renata Teixeira, Sara Ayoubi, Giulio Grassi

The Challenges of Trace-Driven Wi-Fi Emulation

Mohammad Imran Syed[†,¶,§], Renata Teixeira[†], Sara Ayoubi[†], and Giulio Grassi[†]

[†]INRIA, Paris

[¶]Sorbonne Sciences University, Paris

[§]EIT Digital Master School

{*mohammad.syed, renata.teixeira, sara.ayoubi, giulio.grassi*}*@inria.fr*

ABSTRACT

Wi-Fi link is unpredictable and it has never been easy to measure it perfectly; there is always bound to be some bias. As wireless becomes the medium of choice, it is useful to capture Wi-Fi traces in order to evaluate, tune, and adapt the different applications and protocols. Several methods have been used for the purpose of experimenting with different wireless conditions: simulation, experimentation, and trace-driven emulation. In this paper, we argue that trace-driven emulation is the most favourable approach. In the absence of a trace-driven emulation tool for Wi-Fi, we evaluate the state-of-the-art trace driven emulation tool for Cellular networks and we identify issues for Wi-Fi: interference with concurrent traffic, interference with its own traffic if measurements are done on both uplink and downlink simultaneously, and packet loss. We provide a solid argument as to why this tool falls short from effectively capturing Wi-Fi traces. The outcome of our analysis guides us to propose a number of suggestions on how the existing tool can be tweaked to accurately capture Wi-Fi traces.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

Internet measurement, Wireless link emulation, Wi-Fi record & replay

1. INTRODUCTION

Mobile networks are becoming increasingly more popular than wired networks due to the widespread use of mobile devices (e.g. smartphones, laptops, tablets, smart watches, etc.). The number of smartphone users alone is expected to reach 2.87 billion by 2020 [9]. The portability, availability, affordability and increasing speeds of wireless connections, have made wireless the medium of choice.

The quality of wireless connectivity varies drastically from place to place depending on the coverage. There are a number of factors that affect signal quality or create interference like poor network configuration, old equipment, fluctuating demands of users, router position, congestion and coverage. As many of today's applications and services will be running over Wi-Fi or cellular, it is useful to evaluate the performance of these applications in different wireless networks. For instance, application developers may wish to understand the impact of Wi-Fi packet drops on their application, or what will be the user-perceived latencies over Wi-Fi versus LTE.

There are different options for evaluating the applications and services in real network environments, namely simulation, experimentation and emulation. **Simulation** is the easiest way to experiment with different wireless networks conditions. Simulators are used to mimic the behaviour of a certain network in a software-based environment. They offer different topologies, different network entities likes routers, nodes, access points, etc., tuning of real network parameters like packet loss, jitter, delay, latency. There exist a number of wireless simulation tools [5], [2], [3], [4], [1], [10], [6]. Repeatability, control, configurability and experiments of large scale networks are the advantages of simulation. The main limitation of simulation tools, however, is that they require the user to tune different parameters e.g., level of interference, congestion, loss rate, etc. which may not reflect real wireless network conditions.

At the other end of the spectrum there is **Experimentation**, where developers evaluate their applications over deployed wireless links either over testbeds or by relying on volunteer testers. Results of such experiments capture the impact of real wireless network conditions. One disadvantage of experimentation is that it offers no repeatability. The variability of wireless networks makes it hard to reproduce the results. The results of experimentation are, therefore, hard to interpret and one cannot distinguish the issues with application versus wireless issues.

Finally, **trace-driven emulation** involves recording real wireless traces and later replaying them under emulated network conditions. The clear benefits of trace-driven emulations is its ability to capture real network conditions, and the repeatability of the experiments. One can run the same network conditions several times, which eases application or system debugging, and enables comparative analysis of different applications or protocols over the same network conditions.

While there exist trace-driven emulation tools for cellular [18] and web traffic [14], to the best of our knowledge, there exist no such tools for Wi-Fi. In this paper, we evaluate how well the state-of-the-art trace-driven emulation tool [18], originally designed for cellular networks, works for Wi-Fi, and provide suggestions on how such tool can be adapted to correctly record Wi-Fi traces. The rest of this paper is organized as follows: In Section 2, we highlight the existing work. Section 3 explains the challenges in using existing methods. In Section 4, we explain our measurement set-up. Section 5 presents the results of the state-of-the-art trace-driven emulation tool [18] over Wi-Fi. In Section 6, we conclude our work and mention the future work.

2. RELATED WORK

Simulation.

There are several network simulators. NS-2 [5] and NS-3 [2] are open source network simulators to reproduce internet systems. OMNET++ [3] is a simulation platform for building simulators for wireless, wired, queuing networks amongst others. OPNET [4] is an open source network simulation tool which offers various topologies and configurations. NETSIM [1] is a commercial network simulator which provides simulation for layer 1 and layer 2 capabilities of WLAN. QualNet [10] is a commercial network simulator for scalable network technologies. It offers a GUI to make things easier for users as there is no coding involved. TraceReplay is an application layer simulator built in NS-3 for network traces [6]. Despite the fact that there are a lot of simulators available, it is always very hard to get realistic settings.

Testbeds.

Wireless Hybrid Network (WHYNET) [19] is a hybrid testbed as it allows use of simulation, emulation and real hardware. It allows to integrate these on both individual and combined levels. There is limited remote access to WHYNET testbed infrastructure. ORBIT [17] is a radio grid emulator which provides functionality of reproducing wireless experiments with large number of nodes. It allows to introduces fading and controlled interference. MONROE [13] is an open access hardware-based measurement platform for doing experiments on mobile broadband. The advantages of testbeds include running experiments over real wireless links and remote management. However, the testbeds come with a few drawbacks like no repeatability, no mobility (of nodes), small-level scaling and dependency on location.

Emulation.

There are several network emulators that have been previously used to emulate network conditions for Wi-Fi and other technologies. Mobile network tracing [15] observes traffic passively to generate traces and then uses Packet Modulator (PaM) to corrupt, delay or drop captured packets. However mobile network tracing does not address the question of different machines sharing the same bandwidth. Trace-modulation [16] listens to a path passively multiple times to generate traces of real network behaviour. Common Open Research Emulator (CORE) [7] is a network emulator that boasts a GUI which helps in drawing topologies. While CORE emulates layer 3 and above, Extendable Mobile Ad-hoc Network Emulator (EMANE) [11] emulates physical and data link layers (1 and 2). Mahimahi[14] is a framework for recording and replaying HTTP traffic under different network conditions. Mahimahi uses DelayShell for emulating a fixed propagation delay and LinkShell for emulating fixed and variable capacity links. MpShell [8] extends the Mahimahi framework to record Wi-Fi and LTE traces simultaneously. This work was mainly developed to evaluate the performance of MP-TCP in different network conditions and for various types of applications. A mobility emulation framework called EmuWNet [12] is proposed for signal propagation measurements in wireless networks. It allows users to replay measurement traces collected either by simulations or real world experiments. It is based on ORBIT [17] testbed and offers various mobility scenarios for testing in a controlled environment.

In this paper we opt for trace-driven simulation and state-of-the-art trace-driven emulation tool Saturator [18]. Trace-driven emulation is a good option because 1) testing takes place on real network and 2) traces help in repeatability, the results and testing environment can be reproduced later. We, therefore, prefer trace-driven emulation over simulation, testbeds and experimentation.

3. BACKGROUND

In cellular networks, the only form of congestion at the base-station is self-induced congestion. Further, in cellular networks, the uplink and downlink communications of users take place on different time slices and they do not interfere with each other. There are rarely any standing queues created by the traffic of other users in the cell. Even in the case of individual queues, a queuing delay of 750 ms does not starve the load [18]. Whereas the medium is shared in Wi-Fi and hence, the queues at every network entity are shared by all users. Cellular networks are also more robust because there are several number of retransmissions to cope with packet loss which is not the case with Wi-Fi. The throughput can only be affected by the demand and competition for allocation of the time slices in cellular networks whereas there are other factors those affect throughput in Wi-Fi (including cross traffic). This introduces unique challenges for recording Wi-Fi traces.

We used state-of-the-art tool Saturator [18] to demonstrate its behaviour over cellular network and Wi-Fi. Saturator consists of two sender programs running at a client and a server. The client is connected via two cellphones, one cellphone is used to saturate the uplink and the downlink, while the second cellphone is used for feedback. A window of N packets is maintained by each sender program. Using the feedback packets, each sender adjusts the window size to ensure that the link is saturated without causing any self-induced packet loss. Both client and server store in their log the time each data or ACK packet is received, as well as its sequence number, and estimated RTT or 1-way delay. Using these logs, uplink and downlink latency, throughput and packet loss can be computed. The feedback in saturator consists of ACK packets sent to the sender for the packets received by the receiver. The sender can then keep sending consistently to saturate the link reliably. Therefore a separate interface is needed for feedback to ensure timely delivery of ACK packets to the sender; and to avoid any impact of feedback delay on the link saturation. If the interface that has to be saturated is also used for feedback, queuing might cause enough delay for ACK packets to arrive on time. In this case there is a possibility the link might not get properly saturated.

The traces collected via Saturator are replayed using Cellsim [18]. Cellsim requires its own machine with two Ethernet interfaces. It is connected to the client directly with an Ethernet cable whereas it is connected to the Internet with the second Ethernet interface. The client machine is not connected to the Internet, whereas the server machine is connected to the Internet via the Ethernet. Cellsim delays the packets received on both its Ethernet interfaces by a considerable amount of time to emulate propagation delay before adding the packets to the queue. The traffic from the client is sent to the server over the internet by Cellsim.

We collected the traces with the Saturator for both LTE and Wi-Fi. We then used Cellsim to replay the traces captured with the Saturator. We set-up 3 machines for replay; one as client, one as server and one as cellsim machine. We measured the time it took to download a given file over a Wi-Fi network; and compared the results against the time it took to download

Network (s)	Traces (s)
0.40	0.40
0.29	0.29
0.32	0.32
0.37	0.37
0.37	0.37

(a) LTE

Network (s)	Traces (s)
0.24	0.32
0.18	0.31
0.20	0.50
0.24	0.30
0.22	0.32

(b) Wi-Fi

Table 1: File transfer completion times in seconds

This table show transfer completion time for downloading a file over LTE and Wi-Fi (Network columns) versus over recorded and replayed LTE and Wi-Fi traces (Traces columns).

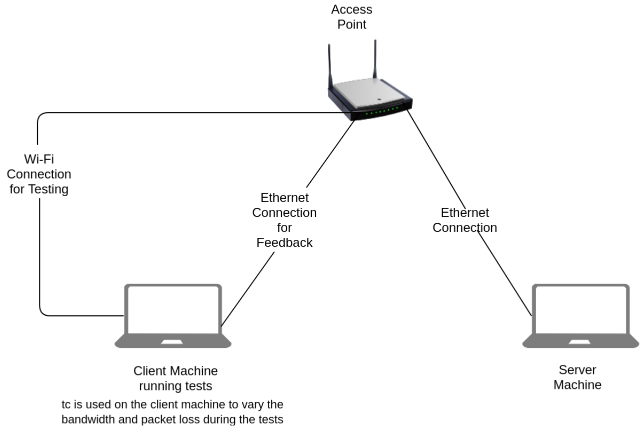


Figure 1: Experiment Set-up

This figure shows the experiment set-up for our tests which includes 2 Dell laptops and a TP-Link access point

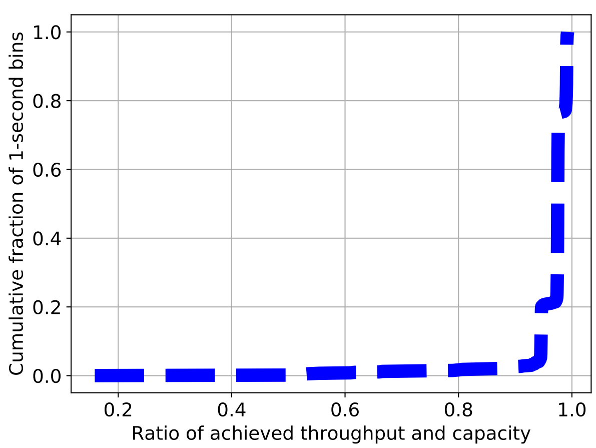


Figure 2: Distribution of the fraction of capacity consumed by the Saturator

the exact same file using traces recorded with the Saturator for the same Wi-Fi network. We did the same for LTE. To ensure that the LTE and Wi-Fi conditions do not fluctuate much between the real and trace-driven experiments, we run the real and emulated experiments back-to-back. We use a 250 MB file for the Wi-Fi experiment, and a 15 MB file for LTE given that we had limited data for LTE.

We can see in Table 1 that the file transfer completion times in both record and replay are always exactly the same for LTE. This was expected because saturator was designed for cellular networks. Whereas for Wi-Fi it took more time to complete the transfer in the replay phase. This indicates that the Saturator works really well for LTE as expected but it is most probably not measuring the Wi-Fi link properly. There is likely to be some error in the Wi-Fi measurements that creates a doubt about the Saturator's compatibility with Wi-Fi.

4. EXPERIMENTAL SET-UP

In this section, we present the experimental set-up that we used for all our measurements. Our measurement set-up, shown in Figure 1, is as follows:

- one laptop as client; connected to Wi-Fi for measurements and connected to an access point with an Ethernet cable for feedback.
- one laptop as server; connected to same access point as the client with an Ethernet cable

The AP was TP-Link Archer C7 which supports 802.11ac. It supports 2.4 GHz and 5 GHz connections simultaneously. We did our testing with 802.11ac and 2.4 GHz. We used 2 Dell laptops with identical specifications; Intel 8th generation core i7 CPU - 1.9GHz (Turbo 4GHz), 16 GB RAM and 520 GB SSD hard drive. Both machines had Ubuntu 18.04 freshly installed, they did not have anything else installed on them. We used this set-up to avoid any impact of CPU load on the Saturator.

We ran saturator client and server versions on these machines. We used Linux's traffic control (tc) option on the client machine to vary the bandwidth and loss rate. We needed to limit the bandwidth to a certain value for testing saturator's compatibility with Wi-Fi. We varied the bandwidth values every 12 seconds as follows: 15 Mbps, 40 Mbps, 10 Mbps, 30 Mbps and 15 Mbps.

Similarly for loss rate, we used tc and varied the percentage value for packet loss every 12 seconds as follows: 0.3%, 0.5%, 0.25%, 1% and 0.3%.

All the tests were performed 5 times each when we had all set-up in place. The test duration was 1 minute for all the tests. We conducted tests for constant and variable bandwidth and packet loss values.

5. SATURATOR OVER WI-FI

In this section, we showcase the limitations of Sat-

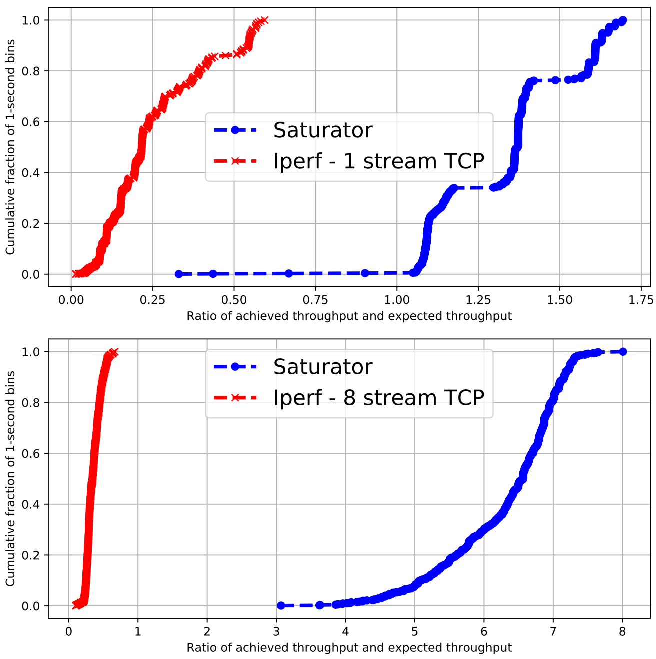


Figure 3: Saturator with concurrent TCP traffic and variable bandwidth

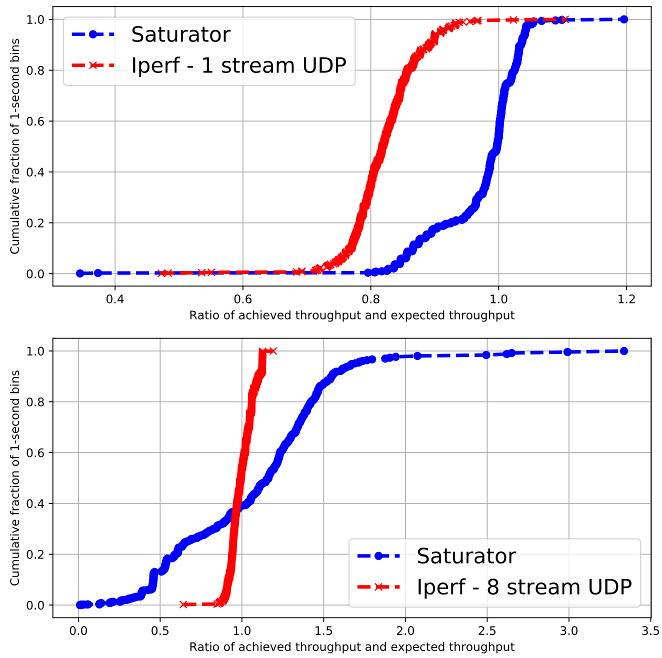


Figure 4: Saturator with concurrent UDP traffic and variable bandwidth

The effect of saturator on concurrent UDP traffic in presence of variable bandwidth is represented in this figure

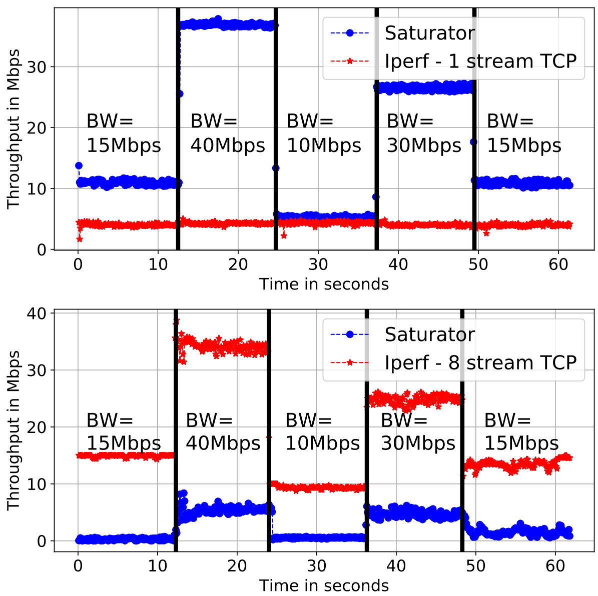


Figure 5: Saturator with concurrent UDP traffic and variable bandwidth

BW means bandwidth and it represents the bandwidth limit set by tc

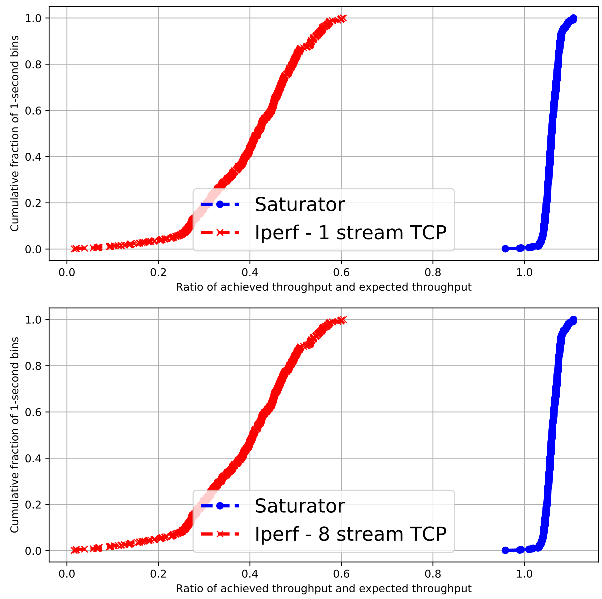


Figure 6: Saturator with concurrent TCP traffic and variable packet loss

urator for Wi-Fi. Given that Wi-Fi's queuing mechanism and medium access control are different than that of Cellular, we study the impact of these two features on the Saturator's ability to accurately record Wi-Fi traces. First, we evaluated Saturator's ability to saturate the Wi-Fi link without any concurrent traffic, a setup that closely resembles Cellular links. Next, we evaluated the behaviour of Saturator when we introduced concurrent UDP and TCP traffic on the Wi-Fi link. Finally, we looked at the impact of saturating both the Wi-Fi uplink and downlink simultaneously.

5.1 Saturator with Concurrent Traffic

The first step was to test Saturator with Wi-Fi without any modifications and in the absence of cross traffic; to verify if the saturator was able to saturate the link. We made use of tc to limit the bandwidth as explained in the previous section and carried out tests without any concurrent traffic.

We observed that Saturator was able to fill the pipe. Saturator reacted to the bandwidth variations very well and adapted accordingly. This was the expected behaviour because the testing conditions without concurrent traffic are similar to cellular networks. Figure 2 shows the ratio of achieved throughput and capacity. The more ratio is closer to 1, the more the Saturator is able to fill the pipe. Smaller values of the ratio in Figure 2 can deceptive because the Saturator was able to consistently saturate the link. These small values of ratio, however, are the result of changing bandwidth during the tests as Saturator takes nearly 1-2 milliseconds to adapt to the new value.

The main question was how would the saturator cope with concurrent traffic in Wi-Fi. We considered the following conditions while doing these tests:

- with concurrent traffic and variable bandwidth
- with concurrent traffic and variable packet loss

We generated concurrent TCP and UDP traffic with IPerf; and we limited the per-IPerf stream bandwidth to 5 Mbps. We can see in Figure 3 that the ratio between achieved throughput and expected throughput is mostly close to 1 for the Saturator; it did considerably well to fill the pipe with concurrent TCP traffic even with variable bandwidth. However it ended up suppressing everything else as TCP traffic got a lot less than what was expected. As evident from Figure 3, the ratio for IPerf is way less than 1. IPerf got around 1 Mbps for 1 stream, whereas it got a maximum of 10 Mbps for 8 streams. It shows Saturator is not fair to TCP traffic.

The results were, however, different for concurrent UDP traffic generated by IPerf. When we used just 1 stream restricted to 5 Mbps, IPerf consistently managed to achieve the expected throughput. However, as we increased the number of streams for UDP traffic, IPerf's UDP traffic seemed to saturate the pipe completely, as we can observe in Figure 4. However, we can see in Figure 4 that the ratio of achieved throughput and expected throughput for the Saturator even exceeds 1; which suggests the Saturator achieved the expected throughput. Figure 5 clears this anomaly; we can see that the Saturator is able to achieve the ex-

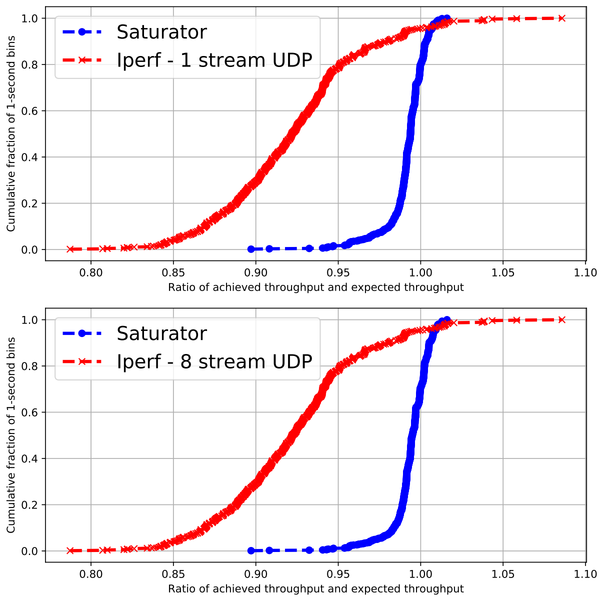


Figure 7: Saturator with concurrent UDP traffic and variable packet loss

pected throughput for higher bandwidth values (i.e. 40 Mbps and 30 Mbps). However, IPerf UDP dominated for lower values of bandwidth (i.e. 15 Mbps and 10 Mbps). This raises a question of Saturator's compatibility with Wi-Fi. Figure 5 shows result of one of the experiments but the results were consistent across all experiments. As discussed earlier, concurrent traffic is very likely to be present and Wi-Fi is not a stable medium as well, it casts doubt on the use of Saturator.

Another important aspect to study about Saturator was its ability to react to the packet loss. We used tc to introduce packet loss as explained in Section 4.

We observed Saturator was able to cope with the packet loss reasonably well as compared to the concurrent traffic. It is evident from Figures 6 and 7 that it keeps sending more traffic to reach a decent throughput as the ratio between achieved throughput and expected throughput is consistently close to 1. It is a surprising finding considering 1) it is UDP traffic and 2) saturator does not react to packet loss itself. We decided to examine it further to find the cause of this behaviour and analysed the logs. As mentioned in Section 3 saturator reacts to RTT only, it just keeps the check of number of packets in flight with respect to the window size and keeps sending packets whenever there is an opportunity. As packets get lost, there are less packets in flight and saturator sees it as an opportunity to send more. In this way it ends up sending more packets than it normally does in case of no loss. The sender side has number of packets sent more than the number of packets received at the receiver by a certain percentage in accordance with the loss.

5.2 Saturator in One Direction

As mentioned earlier, the Saturator was originally designed to work with cellular network and it saturated both uplink and downlink at the same time. It did not matter much for cellular networks because both communications take place on different time and frequency slots and they do not affect each other. Whereas Wi-Fi is a shared medium and there are always chances that uplink and downlink communications happening at the same time can interfere with each other resulting in decreased performance. We therefore made a minor change in Saturator to make it work in only one direction. We evaluated Saturator with Wi-Fi for 2 conditions as following:

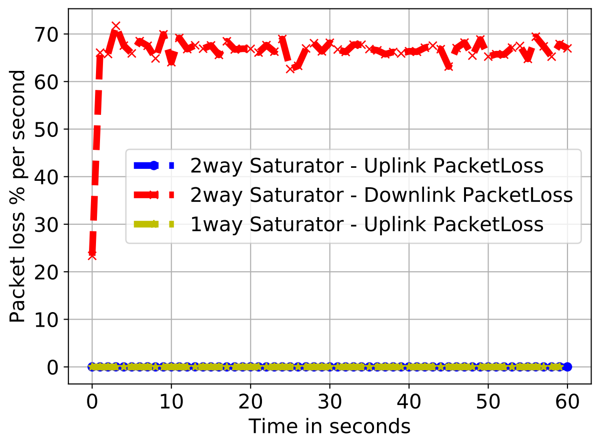


Figure 8: Saturator Packet Loss

This figure is represents the packet loss calculated from saturator logs

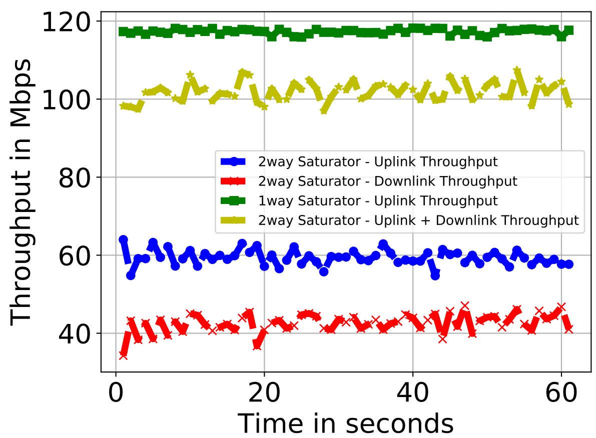


Figure 9: Saturator Throughput

This figures shows comparison of 2-way saturator throughput versus 1-way saturator throughput

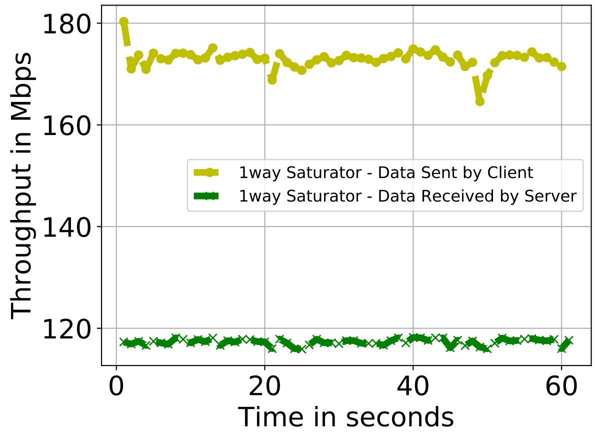


Figure 10: Data sent vs data received - 2-way Saturator

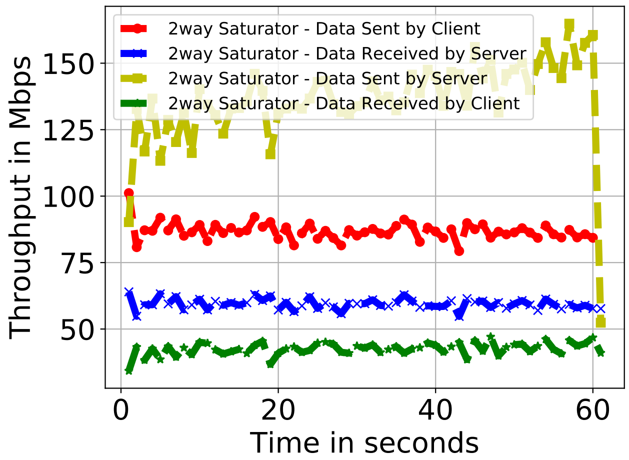


Figure 11: Data sent vs data received - 1-way Saturator

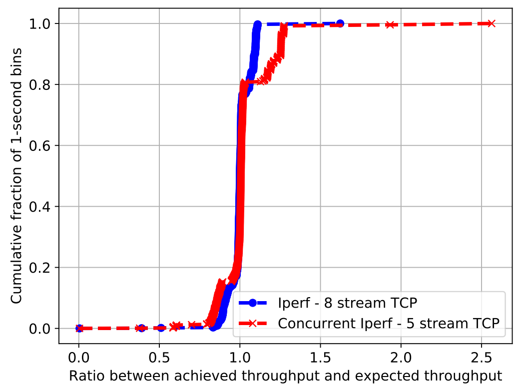
- saturate both uplink and downlink
- saturate just uplink

In the two-way experiments where Saturator is run in both uplink and downlink directions, it always got more throughput on the uplink. Downlink communication was badly affected by packet loss as packet loss went up to 80%. Figure 8 represents the packet loss for one of the five experiments, it was consistent across all five runs. Uplink packet loss for 2-way saturator is not clearly visible in the figure because it is exactly the same as 1-way uplink packet loss. The client always initiates the communication, which could explain why the uplink achieved more bandwidth than the downlink. We made a slight change to make sure the client just initiated the connection with the server but it actually started sending data with a delay of 1 second. We found out that the server took over the bandwidth initially but as soon as the client started sending, we saw more traffic on the uplink same as in previous cases. Multiple factors could explain this behaviour; one possible explanation is that the uplink queue is larger than the downlink queue. Another possible reason, is that the access point might be giving higher priority for uplink traffic than downlink traffic.

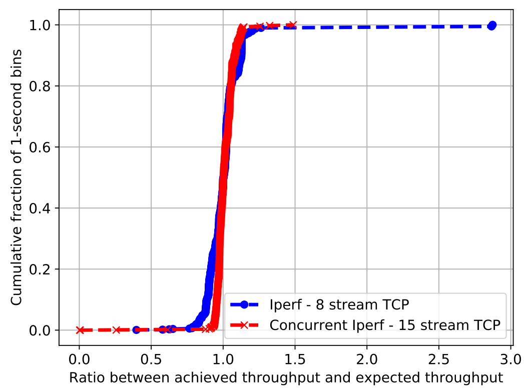
Figure 9 shows the throughputs in Mbps for one run. The results were similar for all runs. We can see that the results were far better and stable when only the uplink is saturated. The 1-way throughput is always more than the sum of uplink and downlink throughputs of 2-way saturator. We recommend to use Saturator only in the uplink direction when used with Wi-Fi to to eliminate the possible interferences and collisions between the uplink and downlink traffic and enable the Saturator to fully saturate the pipe.

5.3 Packet Loss Issue

We discussed earlier there was a huge packet loss with Saturator running in both directions. We used the sequence numbers of packets received to find the packet loss. We were still not fully convinced that Wi-Fi could introduce such losses; so we decided to analyse it further to find out if it was really a Wi-Fi related behaviour.

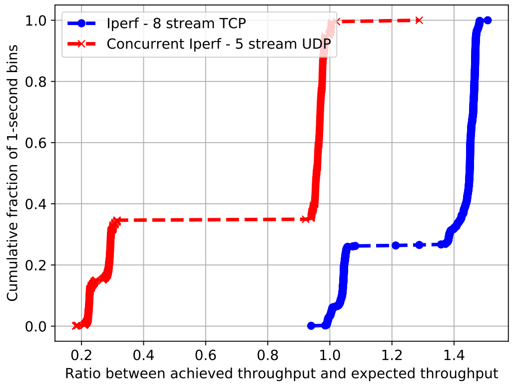


(a) 5 stream TCP Traffic

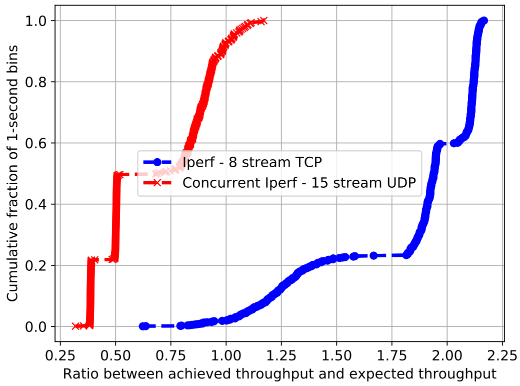


(b) 15 stream TCP Traffic

Figure 12: Multithreaded TCP IPerf - TCP vs TCP



(a) 5 stream UDP Traffic



(b) 15 stream UDP Traffic

Figure 13: Multithreaded TCP IPerf - TCP vs UDP

Since the Saturator only logs incoming data packets at the Sender side, we used tcpdump to be able to capture the client's outgoing data traffic too. We used tshark to get timestamps and packet lengths from the pcap files and then used those to find the throughput. We can see in Figures 10 and 11 that the Saturator sent way more than what was actually received. As mentioned previously Saturator was designed for cellular networks, it did not consider Wi-Fi network conditions and ended up sending more than what it could actually send. This resulted in the driver dropping the packets to cope with excessive Saturator traffic. Therefore, the packet loss we see is not because of Wi-Fi, but it is because the Saturator's sending window was tuned for LTE buffer sizes, which are typically larger than Wi-Fi's.

5.4 Alternative Solutions

Given the aforementioned limitations of the Saturator, in this section, we explore whether IPerf could be used instead to record Wi-Fi traces. We studied the impact of number of threads of TCP IPerf on the bandwidth it measures. We used 8 IPerf TCP streams as traffic generated by us and different number of streams for TCP and UDP IPerf as concurrent traffic. We consider the following scenarios:

- 8 IPerf TCP streams vs 5 IPerf TCP streams
- 8 IPerf TCP streams vs 15 IPerf TCP streams
- 8 IPerf TCP streams vs 5 IPerf UDP streams
- 8 IPerf TCP streams vs 15 IPerf UDP streams

We ran tcpdump in parallel with IPerf and used the pcap files to calculate the throughput. We can see the results for all these scenarios in Figures 12 and 13. The bandwidth measured by IPerf depended on the number of streams used. If IPerf had more streams than concurrent traffic then it ended up getting more bandwidth (mainly because of TCP fair share); that would affect the real traffic in the network and create a bias in the measurements. It shows one has to be careful when choosing number of parallel streams as it can not only create a bias in the measurements but also have negative effect on the actual traffic; especially if multiple streams are used with UDP IPerf, it can badly degrade performance for other users.

6. CONCLUSION

In this paper, we evaluated how well the state-of-the-art trace-driven emulation tool Saturator can capture Wi-Fi traces. Since the Saturator was originally designed for cellular networks, we showcased how the differences between Wi-Fi and Cellular inhibit the applicability of the Saturator for Wi-Fi as-is. We showcased through experimental analysis that the measurements done by the Saturator are influenced by the nature of concurrent traffic. Further, we demonstrated how for Wi-Fi saturating just one direction eliminates interference and improves the ability of the Saturator to effectively fill the pipe.

Yet, all of the experiments we conducted so far were in ideal Wi-Fi conditions, for future work, we aim to repeat our experiments in different setups to study the effects of multipath fading. We would also like to study the impact of 802.11 frame-aggregation on the trace replay. Further, we plan to explore the idea of introducing the packet loss in the replay as this is a common behaviour in Wi-Fi. Finally, we aim to investigate the ideal window size to eliminate the observed packet drops at the driver side.

7. REFERENCES

[1] Netsim simulator by tetcos. `https://www.tetcos.com/wlan.html`.

[2] Ns-3 network simulator. `https://www.nsnam.org/`.

[3] Omnet++ network simulator. `https://omnetpp.org/intro/`.

[4] Opnet network simulator. `http://opnetprojects.com/opnet-network-simulator/`.

[5] Wikipage of ns-2 network simulator. `http://nsnam.sourceforge.net/wiki/index.php/User_Information`.

[6] Agrawal, P., and Vutukuru, M. Trace based application layer modeling in ns-3. In *2016 Twenty Second National Conference on Communication (NCC)* (March 2016), pp. 1–6.

[7] Ahrenholz, J., Danilov, C., Henderson, T. R., and Kim, J. H. Core: A real-time network emulator. In *MILCOM 2008 - 2008 IEEE Military Communications Conference* (Nov 2008), pp. 1–7.

[8] Deng, S., Netravali, R., Sivaraman, A., and Balakrishnan, H. Wifi, lte, or both?: Measuring multi-homed wireless internet performance. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (New York, NY, USA, 2014), IMC '14, ACM, pp. 181–194.

[9] Department, S. R. Number of smartphone users worldwide from 2014 to 2020 (in billions). `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`, 2016.

[10] Dinesh, S., and Sonal, G. Qualnet simulator.

[11] Ivanic, N., Rivera, B., and Adamson, B. Mobile ad hoc network emulation environment. In *MILCOM 2009 - 2009 IEEE Military Communications Conference* (Oct 2009), pp. 1–6.

[12] Karimi, P., Mukherjee, S., Kolodziejski, J., Seskar, I., and Raychaudhuri, D. Measurement based mobility emulation platform for next generation wireless networks. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (April 2018), pp. 330–335.

[13] Midoglu, C., Wimmer, L., Lutu, A., Alay, O., and Griwodz, C. Monroe-nettest: A configurable tool for dissecting speed measurements in mobile broadband networks. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (April 2018), pp. 342–347.

[14] Netravali, R., Sivaraman, A., Das, S., Goyal, A., Winstein, K., Mickens, J., and Balakrishnan, H. Mahimahi: Accurate record-and-replay for http. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference* (Berkeley, CA, USA, 2015), USENIX ATC '15, USENIX Association, pp. 417–429.

[15] Noble, B., Nguyen, G., Satyanarayanan, M., and Katz, R. Mobile network tracing, 1996.

[16] Noble, B. D., Satyanarayanan, M., Nguyen, G. T., and Katz, R. H. Trace-based mobile network emulation. *SIGCOMM Comput. Commun. Rev. 27*, 4 (Oct. 1997), 51–61.

[17] Raychaudhuri, D., Seskar, I., Ott, M., Ganu, S., Ramachandran, K., Kremo, H., Siracusa, R., Liu, H., and Singh, M. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *IEEE Wireless Communications and Networking Conference, 2005* (March 2005), vol. 3, pp. 1664–1669 Vol. 3.

[18] Winstein, K., Sivaraman, A., and Balakrishnan, H. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)* (Lombard, IL, 2013), USENIX, pp. 459–471.

[19] Zhou, J., Ji, Z., Varshney, M., Xu, Z., Yang, Y., Marina, M., and Bagrodia, R. Whynet: A hybrid testbed for large-scale, heterogeneous and adaptive wireless networks. In *Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization* (New York, NY, USA, 2006), WiNTECH '06, ACM, pp. 111–112.